Contents

Pre-requisites

- Follow the Getting Started steps found here [MIMXRT685-EVK Start Now]
- DSP Build environment: Xtensa Xplorer 8.10 + RI-2019.1
- Arm Build environment: MCUXpresso V11.1.1

Objectives

In this lab, you will learn:

- How to configure different audio paths in RT600
  - WM8904 Line-in + WM8904 Line-out
  - DMIC In + WM8904 Line-out
  - USB Audio + WM8904 Line-out

Hardware

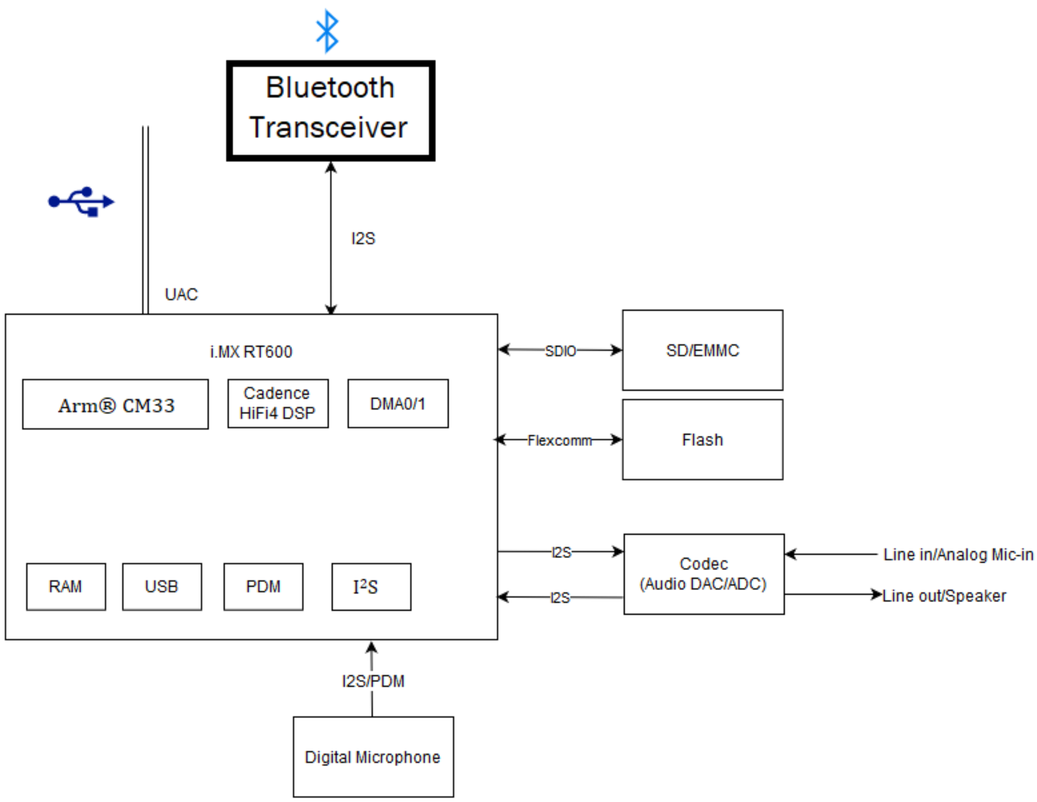
- Micro USB Cable
- MIMXRT685-EVK Rev E.
- Audio source (PC)
- Audio cable 3.5mm
- Headphones with 3.5 mm audio jack
- Female to female jumper wire

Lab high level description

In this lab you will learn how to configure different audio paths in the RT600 based on an initial example configuration. The different audio paths will be configured by the HiFi4 (DSP) and the CM33 will run the USB stack and load the HiFi4 application into RAM and start the HiFi4.

RT600 in an audio system

The audio path is how the audio signal is transferred between the system. In audio applications there are two main components, the audio source and the audio sink. The audio source is where the audio comes from, for example a filesystem (SDcard or eMMC), Bluetooth® transceiver, USB audio class, line-input from codec and digital microphones. The audio sink is the destination of the audio, for example, earpiece, soundbar and headphones. The following diagram shows how these components can interact in an audio application in the RT600:



I²S

The I²S communication interface is used for streaming data transfer applications such as digital audio. The I²S bus specification defines a 3-wire serial bus, having one data, one clock, and one word select/frame trigger signal, providing single or dual (mono or stereo) audio data transfer.

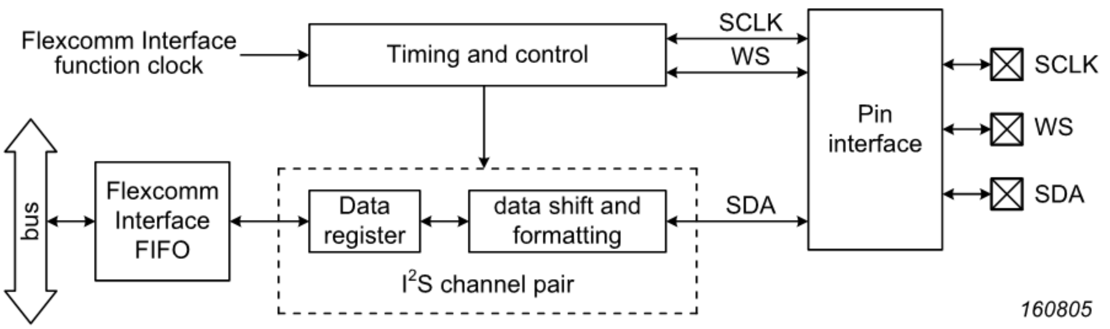
In the RT600, the I²S interface is implemented in selected Flexcomm interfaces. Flexcomm is a serial interface that can be used for UART, SPI, I²C and I²S. In the RT600 there are a total of 8 Flexcomm that support I²S, where 6 of them can support TDM (Time Division Multiplexing).

Flexcomm #	USART	SPI	I2C	I2S
0 - 5	Yes	Yes	Yes	Yes, 4 channel pairs
6 - 7	Yes	Yes	Yes	Yes
14	No	Yes (HS SPI)	No	No
15	No	No	Yes	No

One Flexcomm can only support simplex communication, for full-duplex communication, you need two Flexcomms (one acting as master and one as slave).

I²S Block diagram

The following figure shows a high level diagram of the I²S interface in the RT600. For each interface of I²S there are only three pins (clock pin, word select and data pin).



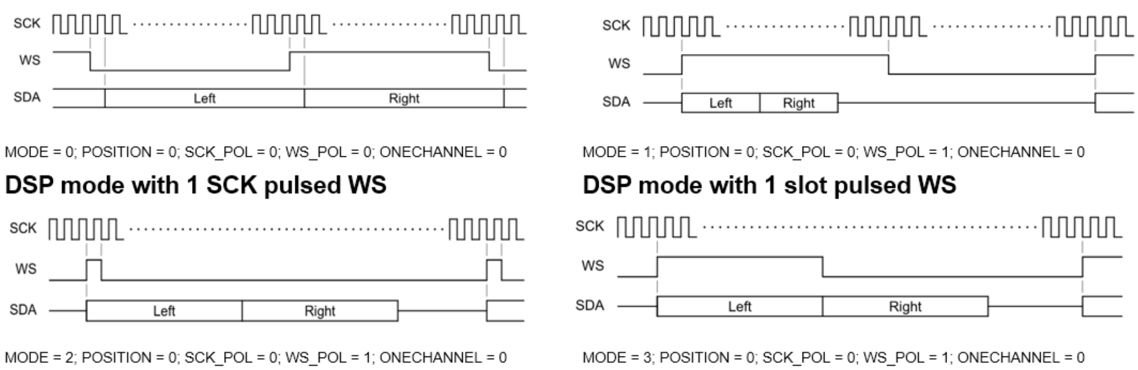
For Flexcomm 0 only, data to be transmitted can optionally be taken directly from DMIC channels 0 and 1.

I²S modes

There are 4 I²S transmission modes, shown in the figure below.

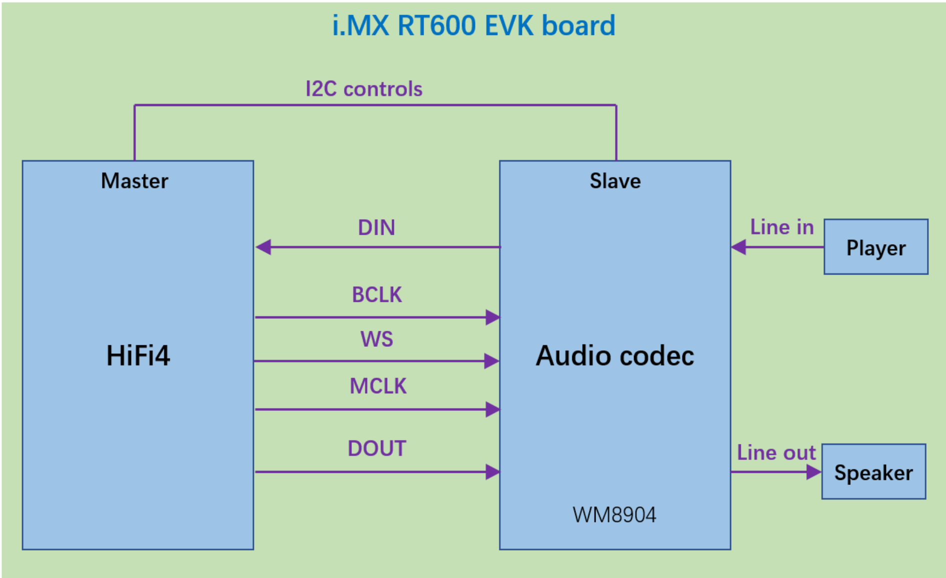
Classic I2S mode

DSP mode with 50% WS



MCLK

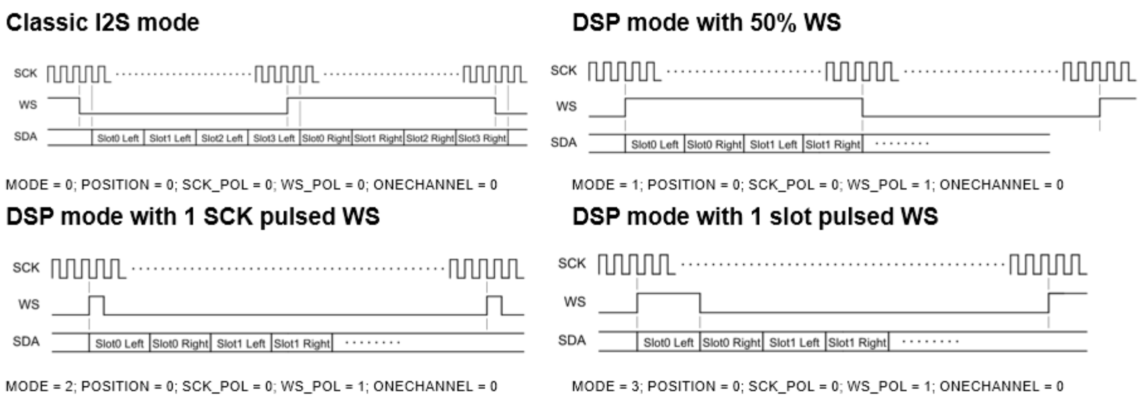
The master clock (MCLK) is another signal shared in some I²S systems. It is used to derive the BCLK and is usually a multiple of 32, 64 and 256 times the sample rate. Audio codecs may use this clock for internal processing so each codec may require a different sample rate multiple. In the RT600, the MCLK can be provided by the Audio PLL or it can be used as a CLK IN when the MCLK is provided by the Audio codec. The following figure shows a typical connection of I²S between the RT600 and an Audio codec. You can refer to the application note AN12749 I2S(Inter-IC Sound Bus) Transmit and Receive on RT600 HiFi4 for more details on I²S on the RT600.



I²S time division multiplexing (TDM)

I²S TDM is a method of using multiple data slots in order to put more channels of data into a single stream. Some new codecs can support I²S TDM so they can use the same 3 I2S pins for

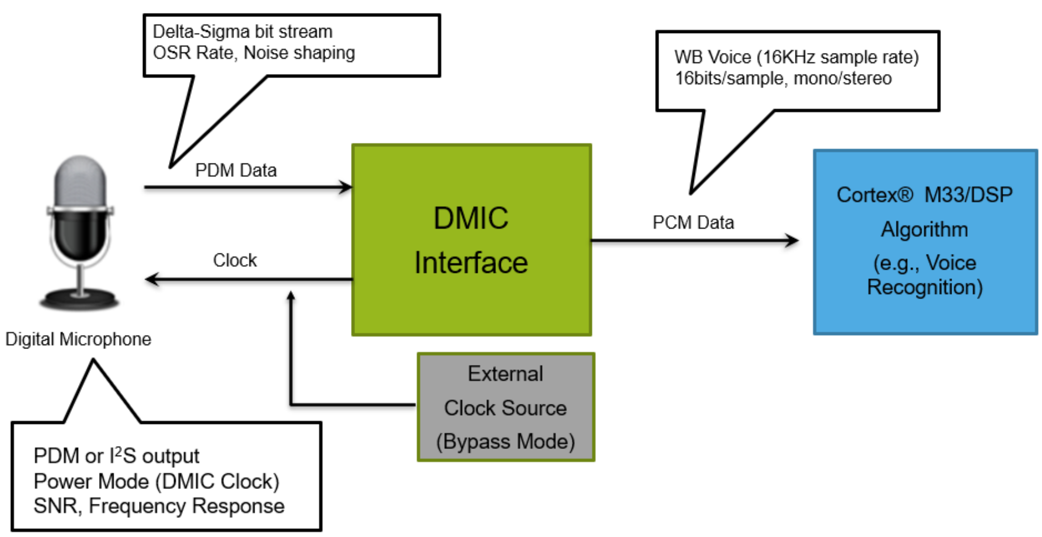
8 channel data. The following figure shows the different I²S TDM modes that can be configured in the RT600.



You can refer to application note [AN12764 8-channel DMIC Audio Acquisition on RT600 HiFi4] for an example of I²S TDM using multiple channels.

DMIC

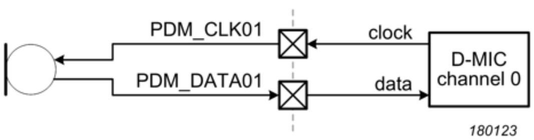
The Digital Microphone Interface (DMIC) in the RT600 can be used to interface with digital microphones that use PDM (Pulse-Density Modulation). The following diagram shows a typical DMIC application:



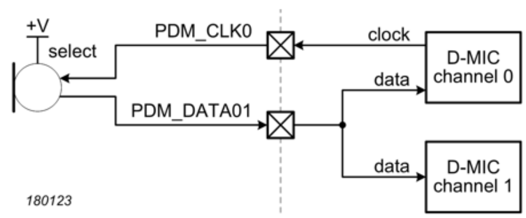
The DMIC interface receives PDM data from multiple digital microphones and processes it to produce 24-bit PCM data. This data can be read by the CPU or DMA, and/or can be sent to Flexcomm0 I²S for output.

DMIC example connections

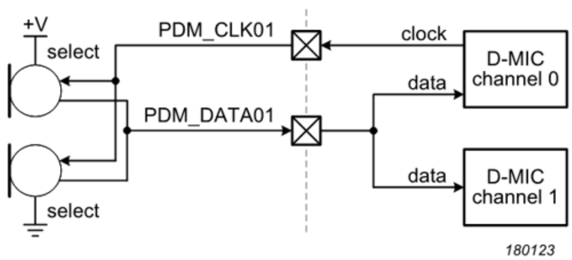
The PDM interface provides options to support 2 single-channel microphones or a single stereo microphone. Specific use examples are shown in the following figure.



connection to a single independent microphone



connection to a stereo microphone



connection to two microphones sharing clock and data lines

The DMIC connection in the MIMXRT685-EVK is two microphones sharing the clock and data lines.

Hardware Voice Activity Detector (HWVAD)

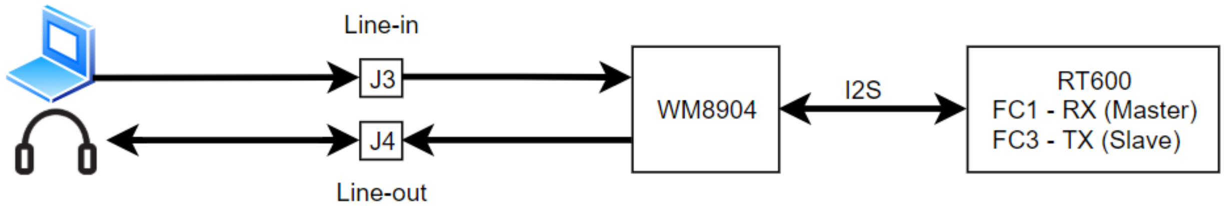
The hardware voice activity detector (HWVAD) is a feature of the DMIC subsystem in the RT600; it implements a wave envelope detector and a floor noise envelope detector. It provides an interrupt when the delta between the two detectors is larger than a predefined value. The input signal for the HWVAD can come from DMIC channel 0. The basic detection of a voice activity can be the starting point for a more sophisticated task, for example, voice recognition.

Audio Lab 1 - WM8904 Line-in + WM8904 Line-out

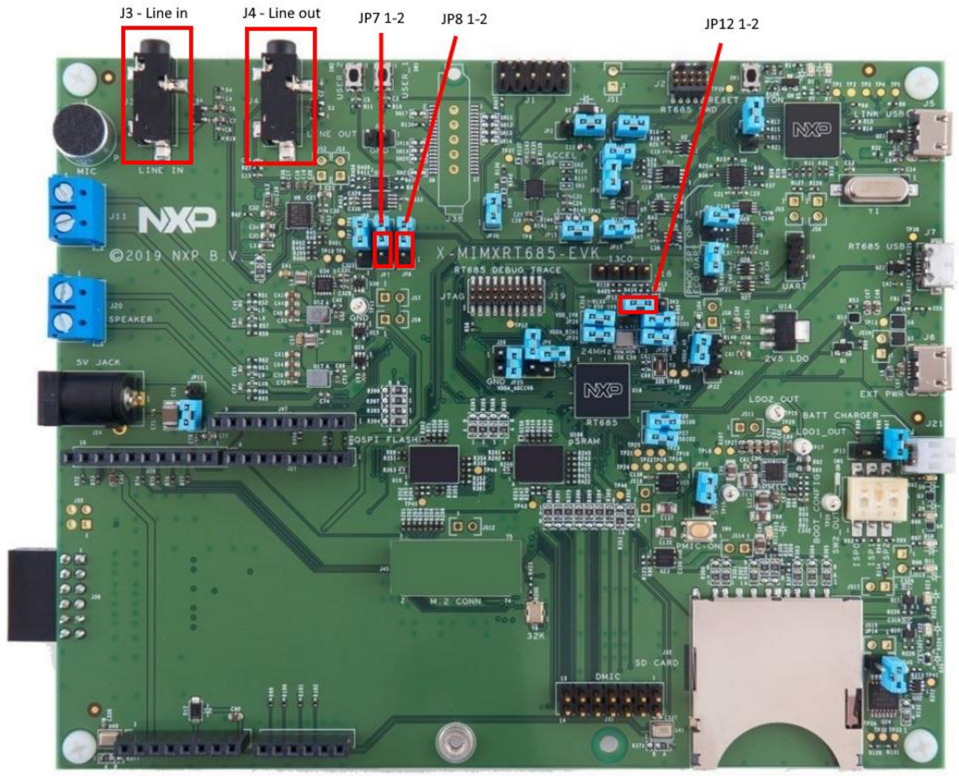
Lab description

For this lab, the audio path is configured to get the audio signal from the external codec (WM8904) Line-in and do a bypass to the Line-out of the codec. The codec handles the analog conversion between its input and output lines and communicates with the RT600 via I^2S.

The I^2S interface in the RT600 is used in full-duplex mode using two I^2S instances with I^2S signal sharing. One instance is used as RX (master) and the other as TX (slave). The following figure shows a high-level diagram of the audio path.



MIMXRT685-EVK settings

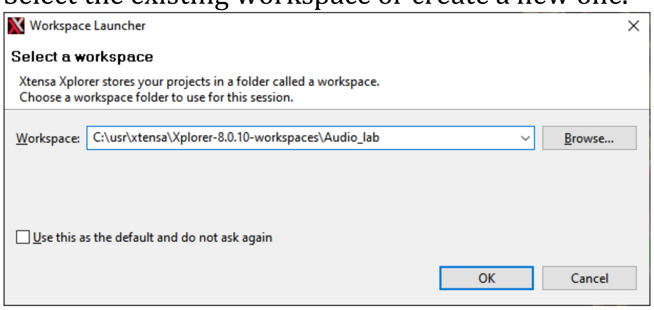


- Connect audio from PC to J3 using audio cable (3.5mm).
- Connect headphones with 3.5 mm audio jack to J4
- Ensure JP12 is connected at 1-2 (1.8V to VDDIO_1)
- Connect JP7 1-2 (I2S TX)
- Connect JP8 1-2 (I2S RX)

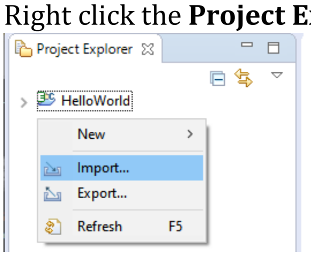
Import HiFi4 project

1. Open **Xplorer 8.0.10**.

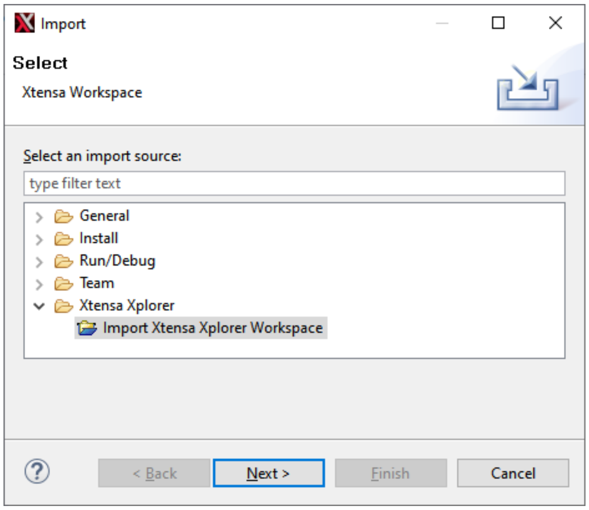
2. Select the existing workspace or create a new one.



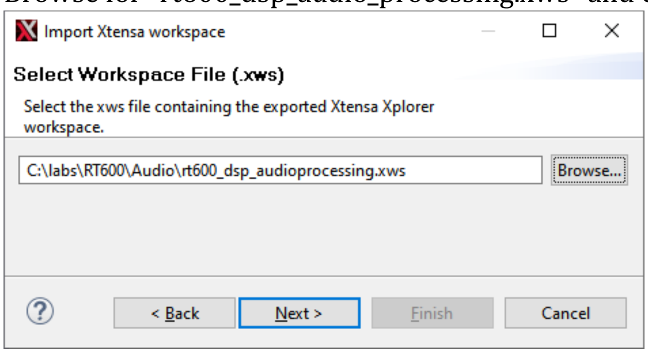
3. Right click the **Project Explorer** window, select **Import…**



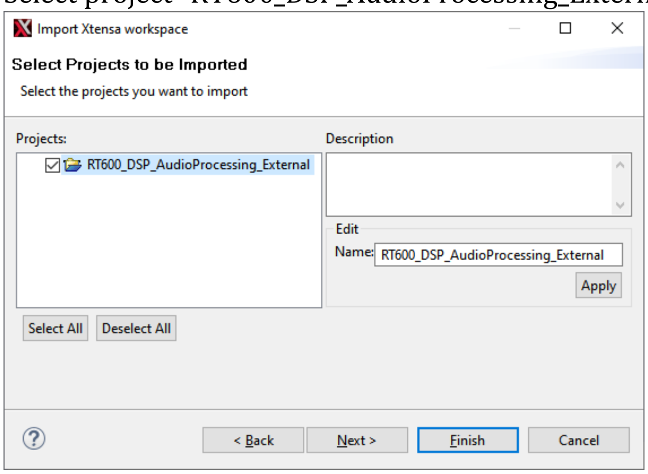
4. Select "**Import Xtensa Xplorer workspace**"



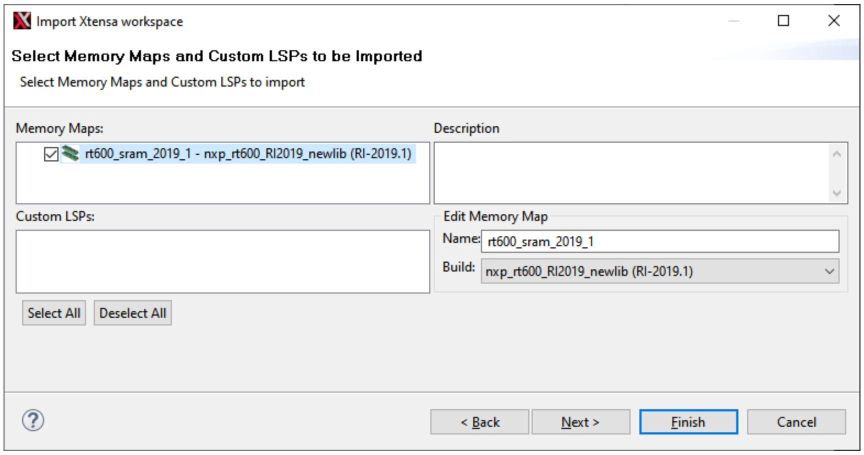
5. Browse for “rt600_dsp_audio_processing.xws” and click **Next**.



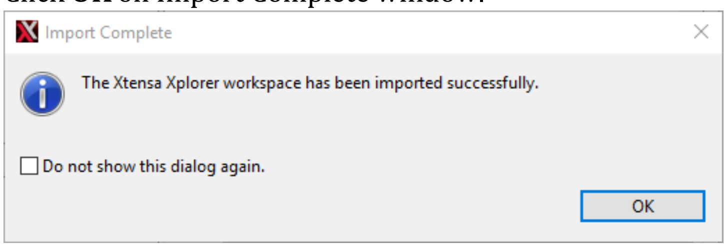
6. Select project “RT600_DSP_AudioProcessing_External” and click **Next.**



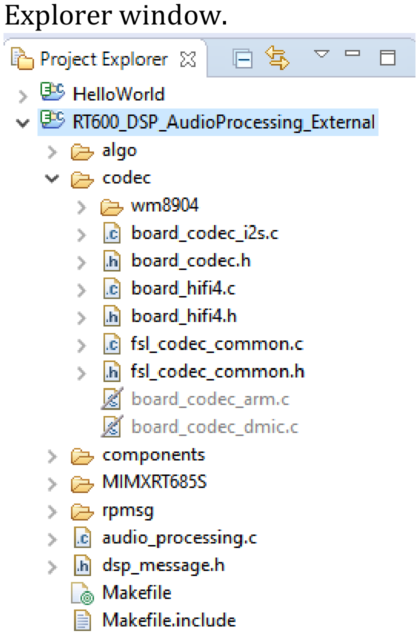
7. Select memory map “rt600_sram_2019_1 –nxp_rt600_RI2019_newlib (RI-2019.1)” and click **Finish.**



8. Click **OK** on Import Complete window.

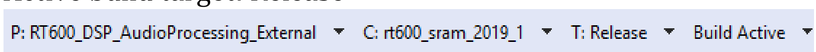


9. The project “RT600_DSP_AudioProcessing_External” will appear in the Project Explorer window.

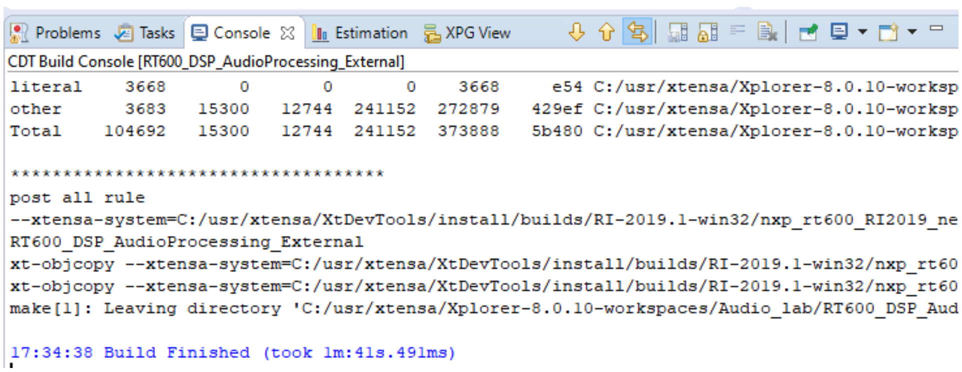


Build HiFi4 project

1. On the menu bar, select the following build configuration:
   - Active project: RT600_DSP_AudioProcessing_External
   - Active configuration: rt600_sram_2019_1
   - Active build target: Release



2. Click on **Build Active** and wait for the build to finish.

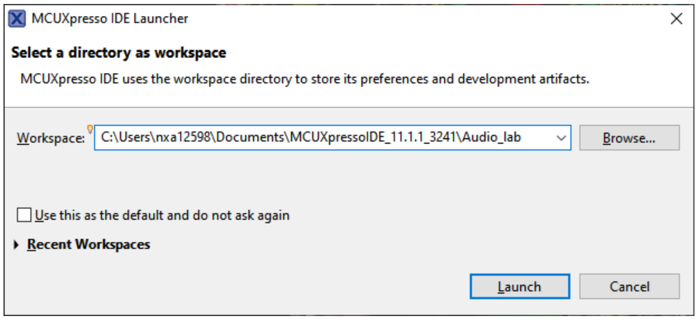


Notice the output binary files "dsp_text_release.bin" and "dsp_data_release.bin" located at *"<workspace>\<project>\bin\rt600_sram2019_1\Release\".* These will need to be copied to the ARM® project in the following steps so that the ARM core can load the DSP image into RAM.

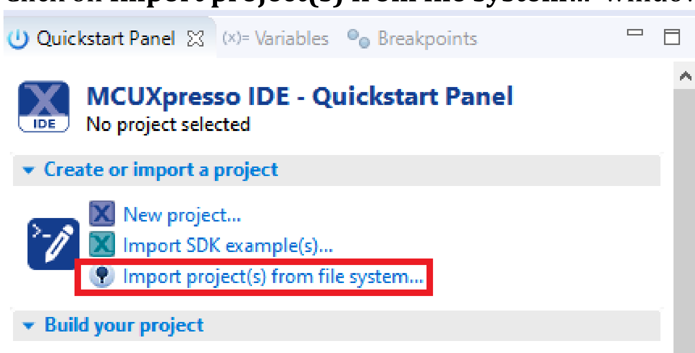
Import ARM project

1. Open **MCUXpresso IDE v11.1.1**.

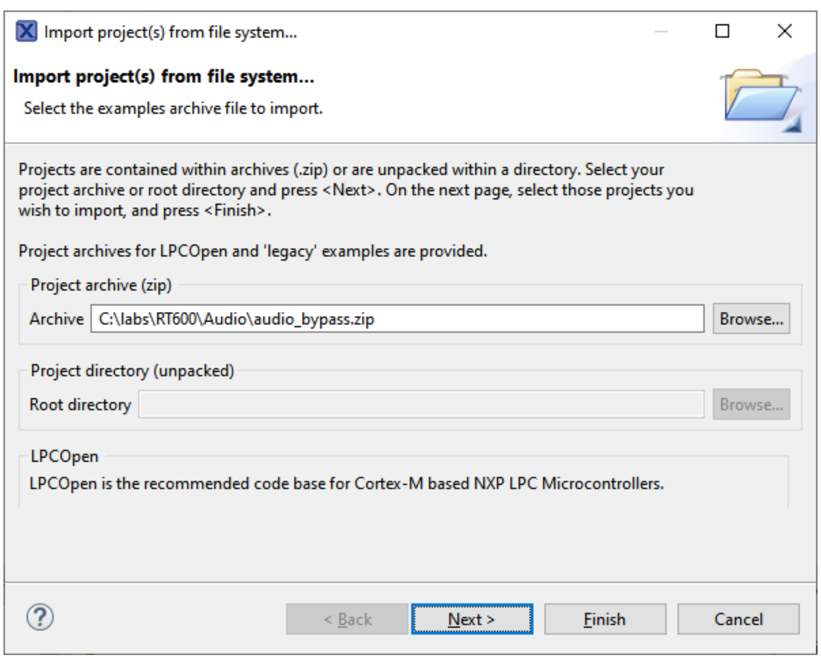
2. Select the existing workspace or create a new one.



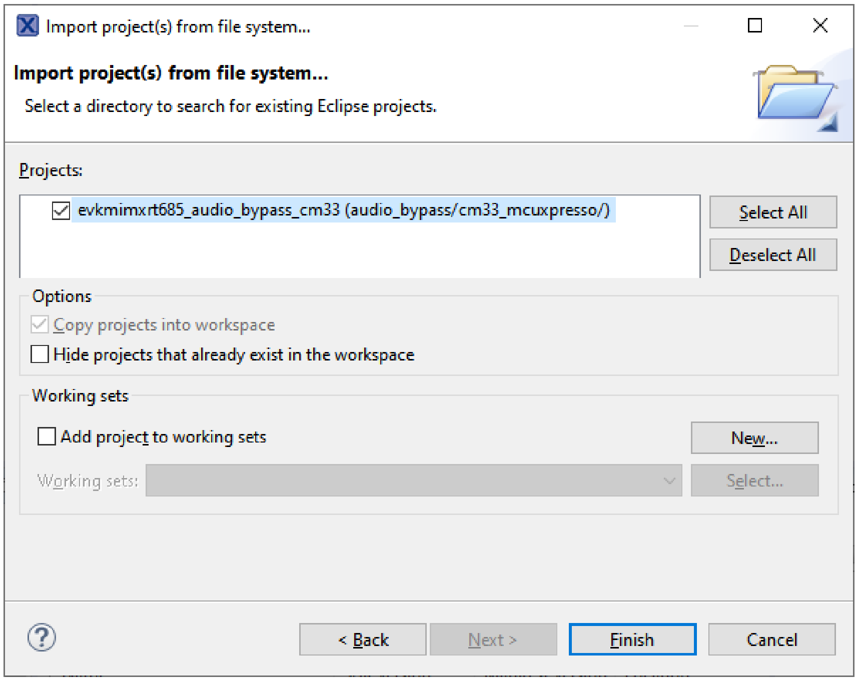
3. Click on **Import project(s) from file system…** window.



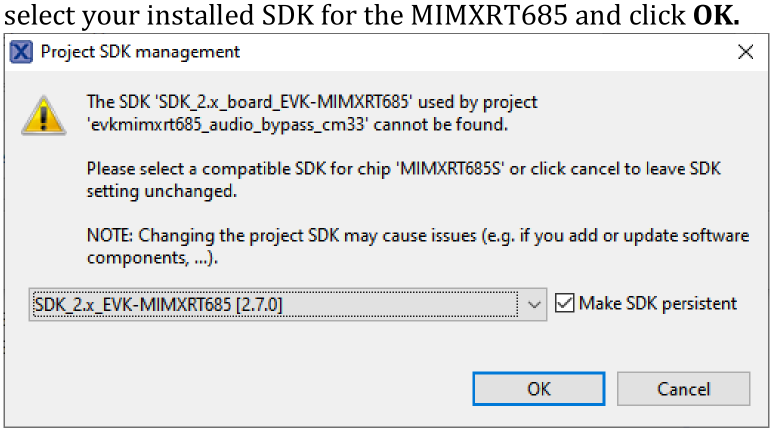
4. In **Project archive(zip)** browse for "audio_bypass.zip" and click **Next**.



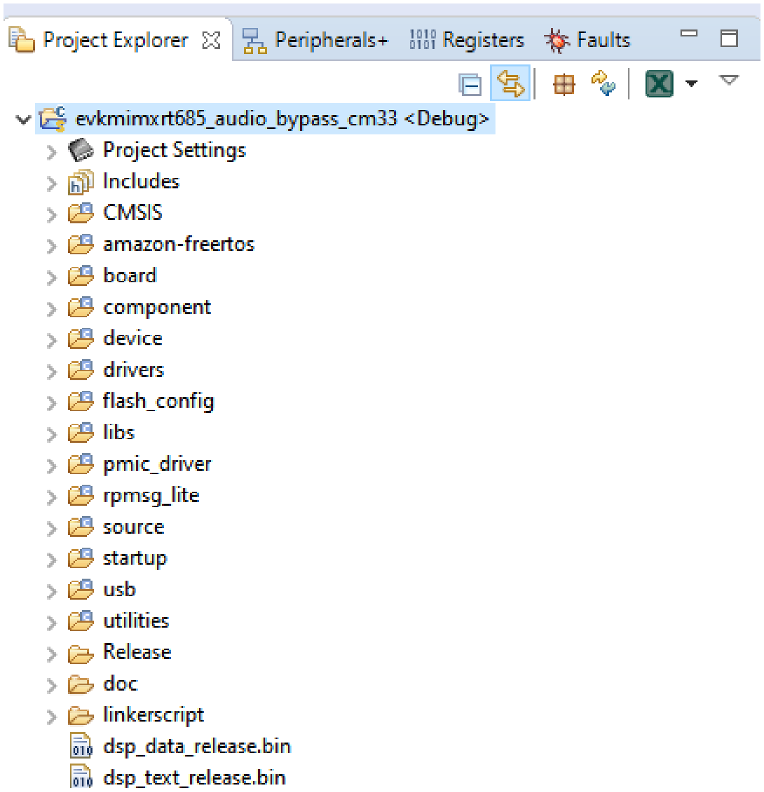
5. Make sure that the "evkmimxrt685_audio_bypass_cm33" project is selected and click **Finish**.



6. If you get a warning window about not finding the exact SDK due to name mismatch, select your installed SDK for the MIMXRT685 and click **OK.**



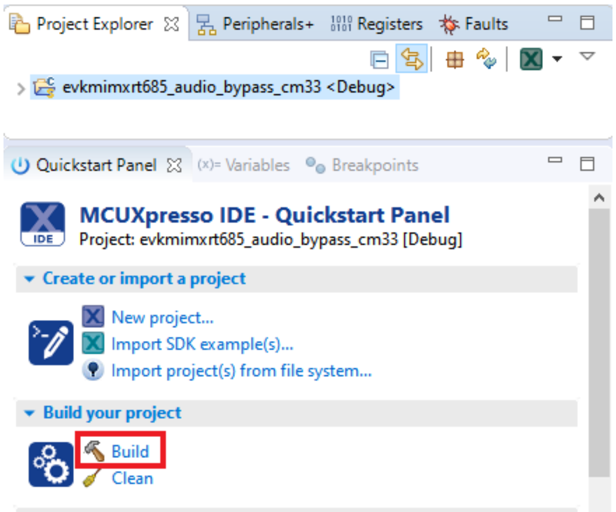
10. The project "evkmimxrt685_audio_bypass_cm33" will appear in the Project Explorer window.



Build ARM project

1. Copy the output binary files “dsp_text_release.bin” and “dsp_data_release.bin” from the HiFi4 project located at *“<workspace>\<project>\bin\rt600_sram2019_1\Release\”* and replace the files located in the ARM project at “<workspace>\evkmimxrt685_audio_bypass_cm33\”.

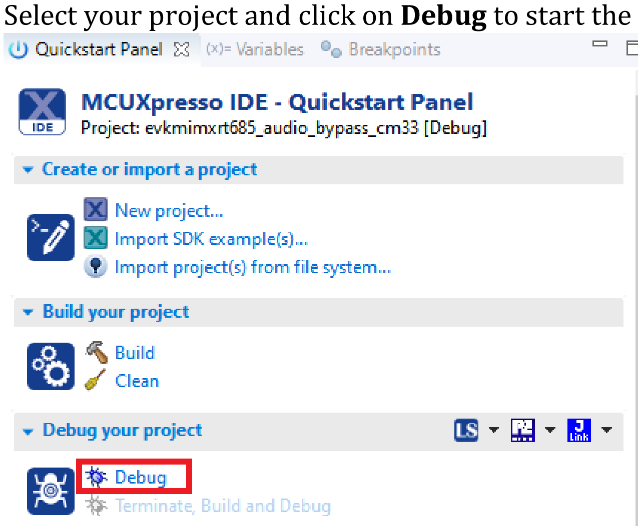
2. Select your project and click on **Build**.



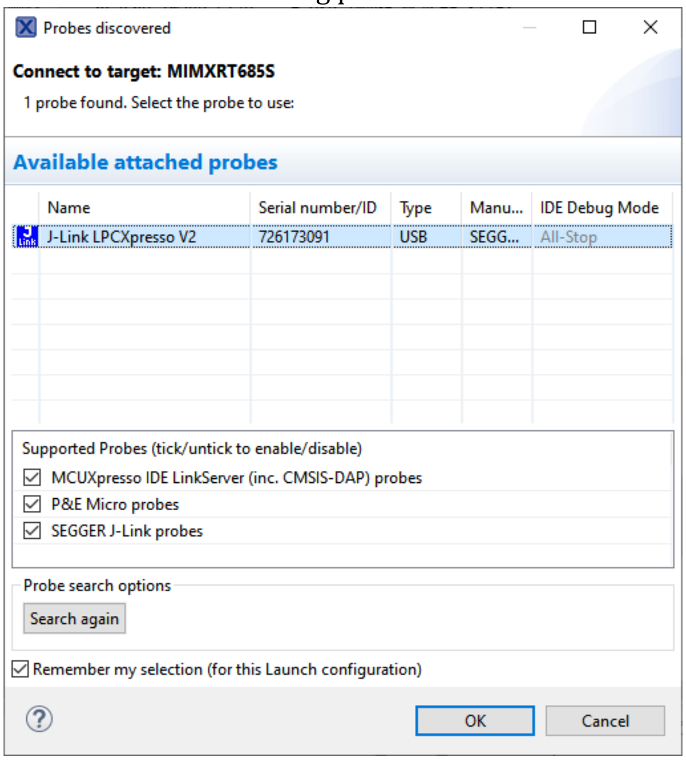
The ARM project now includes the image for the HiFi4 core. It will load the HiFi4 image into RAM and will initialize the HiFi4 to run from the RAM image.

Run the ARM project

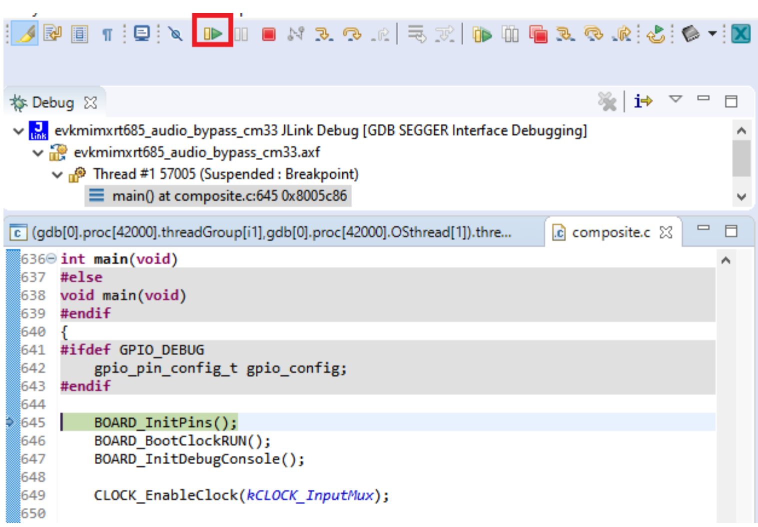
1. Select your project and click on **Debug** to start the debug session.



2. Select the on-board debug probe and click **OK**.



3. The debug session will start. Click on the **Resume** button to start the application.

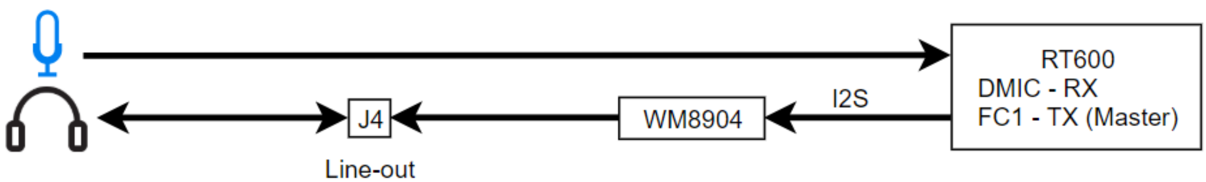


4. Playback an audio file at PC side or audio input connected at J3.
5. You should hear music from the Headphones connected at J4.

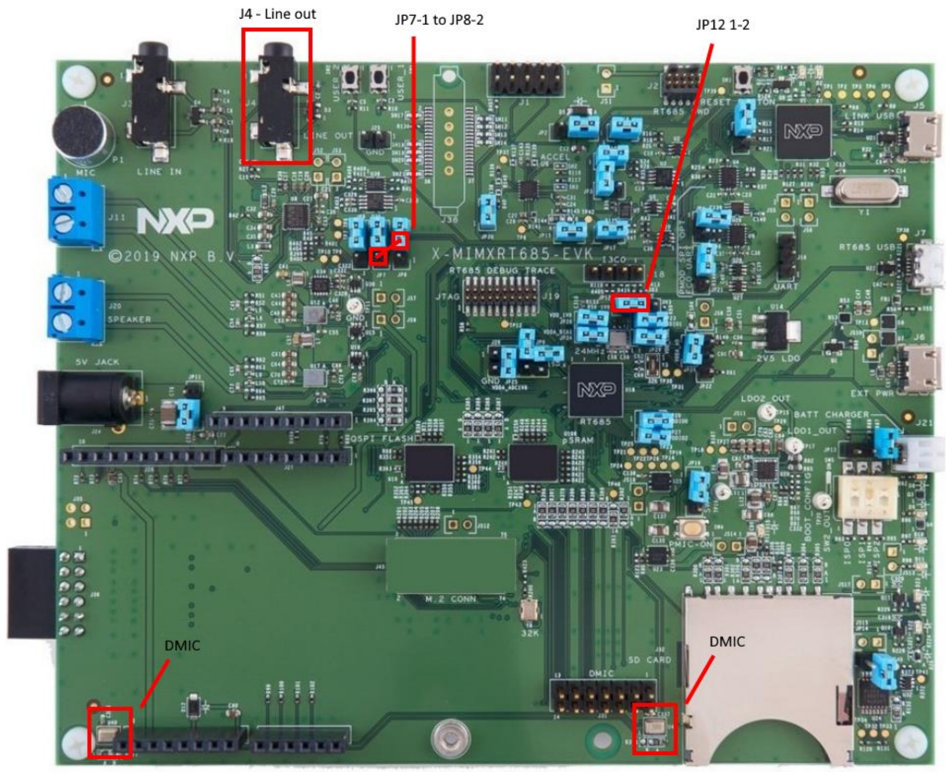
Audio Lab 2 - DMIC in + WM8904 Line-out

Lab description

Lab 2 is based on lab 1, the audio source is changed from the line-in of the codec to the DMIC interface using an onboard digital microphone. The following figure shows a high-level diagram of the audio path.



MIMXRT685-EVK settings



- Connect headphones with 3.5 mm audio jack to J4
- Ensure JP12 is connected at 1-2 (1.8V to VDDIO_1)
- Connect JP8-2 to JP7-1 with jumper wire (I2S TX)

HiFi4 project changes

In lab 3 only one I2S interface is used as TX and the DMIC is used as the RX.

In "<hifi4 project>\codec\board_codec_i2s.c":

- Add necessary file includes:

```
#include "fsl_dmic.h"
#include "fsl_dmic_dma.h"
#include "fsl_iopctl.h"
#include "ringbuf.h"
```

- DMA request from DMIC :

```
#define DMIC_RX_CHANNEL0 (16)
```

- Remove all I²S RX references and keep I2S TX as I2S1:

```
#define I2S_TX (I2S1)
#define I2S_TX_CHANNEL (3)
```

- Add global variables for DMIC configuration:

```
static dma_handle_t s_dmicRxDmaHandle[AUDIO_RX_CHANNELS];
static dmic_dma_handle_t s_dmicDmaHandle[AUDIO_RX_CHANNELS];
static int32_t dmic_channel_index[AUDIO_RX_CHANNELS];

__attribute__((section("NonCacheable")))SDK_ALIGN(dma_descriptor_t
s_dmaDescriptorPingpong[AUDIO_RX_CHANNELS * 2], 16);
static dmic_transfer_t s_receiveXfer[AUDIO_RX_CHANNELS * 2] =
{
#if (AUDIO_RX_CHANNELS == 1)
                       /* transfer configurations for channel0 */
                       { .data = &audioRecDMABuff[0], .dataWidth = sizeof(int16_t),
.dataSize =
                               AUDIO_RX_TRANSFER_SIZE / AUDIO_RX_CHANNELS *
sizeof(int16_t), .dataAddrInterleaveSize =
                               kDMA_AddressInterleave1xWidth, .linkTransfer =
&s_receiveXfer[1], },

                       { .data = &audioRecDMABuff[AUDIO_RX_TRANSFER_SIZE], .dataWidth =
                               sizeof(int16_t), .dataSize = AUDIO_RX_TRANSFER_SIZE /
AUDIO_RX_CHANNELS * sizeof(int16_t),
                               .dataAddrInterleaveSize = kDMA_AddressInterleave1xWidth,
                               .linkTransfer = &s_receiveXfer[0],
                       },
#elif (AUDIO_RX_CHANNELS == 2)
                       /* transfer configurations for channel0 */
                       { .data = &audioRecDMABuff[0], .dataWidth = sizeof(int16_t),
.dataSize =
                               AUDIO_RX_TRANSFER_SIZE / AUDIO_RX_CHANNELS *
sizeof(int16_t), .dataAddrInterleaveSize =
                               kDMA_AddressInterleave2xWidth, .linkTransfer =
&s_receiveXfer[1], },
```

```
                        { .data = &audioRecDMABuff[AUDIO_RX_TRANSFER_SIZE], .dataWidth =
                                sizeof(int16_t), .dataSize = AUDIO_RX_TRANSFER_SIZE /
AUDIO_RX_CHANNELS * sizeof(int16_t),
                                .dataAddrInterleaveSize = kDMA_AddressInterleave2xWidth,
                                .linkTransfer = &s_receiveXfer[0],
                        },
                        /* transfer configurations for channel0 */
                        { .data = &audioRecDMABuff[1], .dataWidth = sizeof(int16_t),
                                .dataSize = AUDIO_RX_TRANSFER_SIZE / AUDIO_RX_CHANNELS *
sizeof(int16_t),
                                .dataAddrInterleaveSize = kDMA_AddressInterleave2xWidth,
                                .linkTransfer = &s_receiveXfer[3], },

                        { .data = &audioRecDMABuff[AUDIO_RX_TRANSFER_SIZE + 1], .dataWidth
=
                                sizeof(int16_t), .dataSize = AUDIO_RX_TRANSFER_SIZE /
AUDIO_RX_CHANNELS * sizeof(int16_t),
                                .dataAddrInterleaveSize = kDMA_AddressInterleave2xWidth,
                                .linkTransfer = &s_receiveXfer[2], },
#else
#error "To be implemented"
#endif
};
```

- Remove **I²SRxCallback()** and add **dmic_dma_Callback()**, in this callback, the audio buffer is updated with the data from the DMIC:

```
void dmic_dma_Callback(DMIC_Type *base, dmic_dma_handle_t *handle,
                status_t status, void *userData) {
        int32_t ch = *(int32_t *) userData;
        AudioRxDMA_Queue_Element element;

        if (ch == AUDIO_RX_CHANNELS - 1) {
                element.pBufAddress = &audioRecDMABuff[AUDIO_RX_TRANSFER_SIZE *
audio_rx_buffer_index];
                element.uBufSize = AUDIO_RX_TRANSFER_SIZE * sizeof(int16_t); /* uBufSize
in Bytes */
                xos_msgq_put(p_audio_rx_dma_queue, (uint32_t *)&element);
                audio_rx_buffer_index ^= 1;
        }
}
```

- It may be necessary to raise the volume of the codec to be able to listen to the ambient sound. In **BOARD_Codec_Init()**:

```
ret = WM8904_SetVolume(&codecHandle, 0x0020, 0x0020);
```

- In **Audio_TxInit()**, keep the Flexcomm 1 configuration and configure I²S as master.
- Replace **Audio_RxInit()**, with the following code, here the DMIC initialization is done:

```
void Audio_RxInit(void) {
        dmic_channel_config_t dmic_channel_cfg;
        int32_t dmic_enable = 0;
        int32_t i;
        bool use2fs = true;

        p_audio_rx_dma_queue = malloc(
                        XOS_MSGQ_SIZE(AUDIO_RX_DMA_QUEUE_LENGTH,
                                sizeof(struct _AudioRxDMA_Queue_Element)));
```

```
        xos_msgq_create(p_audio_rx_dma_queue, AUDIO_RX_DMA_QUEUE_LENGTH,
                sizeof(struct _AudioRxDMA_Queue_Element), XOS_MSGQ_WAIT_PRIORITY);

        /* Initialize DMIC pins below */
        const uint32_t port2_pin16_config = (IOPCTL_FUNC1 | /* Pin is configured
as pdm_clk01 */
        IOPCTL_INBUF_EN /* Enables input buffer */
        );
        IOPCTL_PinMuxSet(IOPCTL, 2, 16, port2_pin16_config); /* PORT2 PIN16 is
configured as DMIC CLK01 */

        const uint32_t port2_pin20_config = (IOPCTL_FUNC1 | /* Pin is configured
as pdm_data01 */
        IOPCTL_INBUF_EN /* Enables input buffer */
        );
        IOPCTL_PinMuxSet(IOPCTL, 2, 20, port2_pin20_config); /* PORT2 PIN20 is
configured as DMIC DATA01 */

        CLOCK_AttachClk(kAUDIO_PLL_to_DMIC_CLK);
        CLOCK_SetClkDiv(kCLOCK_DivDmicClk, 8); //24.576/8= 3.072MHz

        DMIC_Init(DMIC0);
//      DMIC_SetIOCFG(DMIC0, kDMIC_PdmDual);
        DMIC_Use2fs(DMIC0, use2fs);

        memset(&dmic_channel_cfg, 0U, sizeof(dmic_channel_config_t));
        dmic_channel_cfg.divhfclk = kDMIC_PdmDiv1;
        dmic_channel_cfg.osr = 3072 / AUDIO_RX_SAMPLERATE_MS;
        if (use2fs) {
                dmic_channel_cfg.osr = (dmic_channel_cfg.osr >> 1);
        } else {
                dmic_channel_cfg.osr = (dmic_channel_cfg.osr >> 2);
        }
        dmic_channel_cfg.gainshft = 3U;
        dmic_channel_cfg.preac2coef = kDMIC_CompValueZero;
        dmic_channel_cfg.preac4coef = kDMIC_CompValueZero;
        dmic_channel_cfg.dc_cut_level = kDMIC_DcCut155;
        dmic_channel_cfg.post_dc_gain_reduce = 1U;
        dmic_channel_cfg.saturate16bit = 1U;
        dmic_channel_cfg.sample_rate = kDMIC_PhyFullSpeed;

        for (i = 0; i < AUDIO_RX_CHANNELS; i++) {
                DMIC_EnableChannelDma(DMIC0, (dmic_channel_t) (kDMIC_Channel0 +
i), true);
                if ((i % 2) == 0)
                        DMIC_ConfigChannel(DMIC0, (dmic_channel_t) (kDMIC_Channel0
+ i),
                                kDMIC_Left, &dmic_channel_cfg);
                else
                        DMIC_ConfigChannel(DMIC0, (dmic_channel_t) (kDMIC_Channel0
+ i),
                                kDMIC_Right, &dmic_channel_cfg);

                DMIC_FifoChannel(DMIC0, kDMIC_Channel0 + i, 15, true, true);

                dmic_enable |= (1 << i);
```

```
    }
 DMIC_EnableChannnel(DMIC0, dmic_enable);

}
```

- In **BOARD_DMA_EDMA_Config()** remove the references to the I[2]S RX channel and add the DMIC DMA configuration:

```
    int32_t i;
    for (i = 0; i < AUDIO_RX_CHANNELS; i++) {
            dmic_channel_index[i] = i;
            DMA_EnableChannel(DMA1, DMIC_RX_CHANNEL0 + i);
            DMA_SetChannelPriority(DMA1, DMIC_RX_CHANNEL0 + i,
kDMA_ChannelPriority2);
            DMA_CreateHandle(&s_dmicRxDmaHandle[i], DMA1, DMIC_RX_CHANNEL0 +
i);
            DMIC_TransferCreateHandleDMA(DMIC0, &s_dmicDmaHandle[i],
                    dmic_dma_Callback, &dmic_channel_index[i],
&s_dmicRxDmaHandle[i]);
            DMIC_InstallDMADescriptorMemory(&s_dmicDmaHandle[i],
                    &s_dmaDescriptorPingpong[2 * i], 2U);

    }
```

- In **BOARD_DMA_EDMA_Start()** remove the references to the I[2]S RX channel and add the DMIC DMA code for starting:

```
int32_t i;
for (i = 0; i < AUDIO_RX_CHANNELS; i++) {
DMIC_TransferReceiveDMA(DMIC0, &s_dmicDmaHandle[i],
&s_receiveXfer[2 * i], kDMIC_Channel0 + i);
    }
```

Reference code

The changes from the previous step can also be enabled by changing the build configuration in the Hifi4 project as follow:

- Right click “board_codec_i2s.c”, then choose “**Build->Exclude**”
- Right click “board_codec_dmic.c”, then choose “**Build-> Include**”
- Right click “board_codec_arm.c”, then choose “**Build->Exclude**”

Build HiFi4 and ARM project

1. Rebuild the HiFi4 project. See lab 1 for more detailed instructions.
2. Copy the output binary files “dsp_text_release.bin” and “dsp_data_release.bin” from the HiFi4 project located at *“<workspace>\<project>\bin\rt600_sram2019_1\Release\”* and replace the files located in the ARM project at “<workspace>\evkmimxrt685_audio_bypass_cm33\”.

3. Clean the ARM project to ensure that the new HiFi4 output binary files are considered.

4. Rebuild ARM project. See lab 1 for more detailed instructions.

Run the ARM project

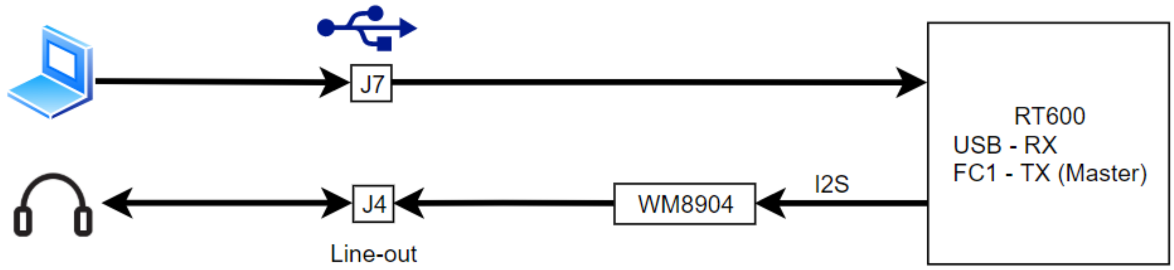
1. Debug and run the arm project.

2. You should hear the onboard microphone audio in the Headphones connected at J4.

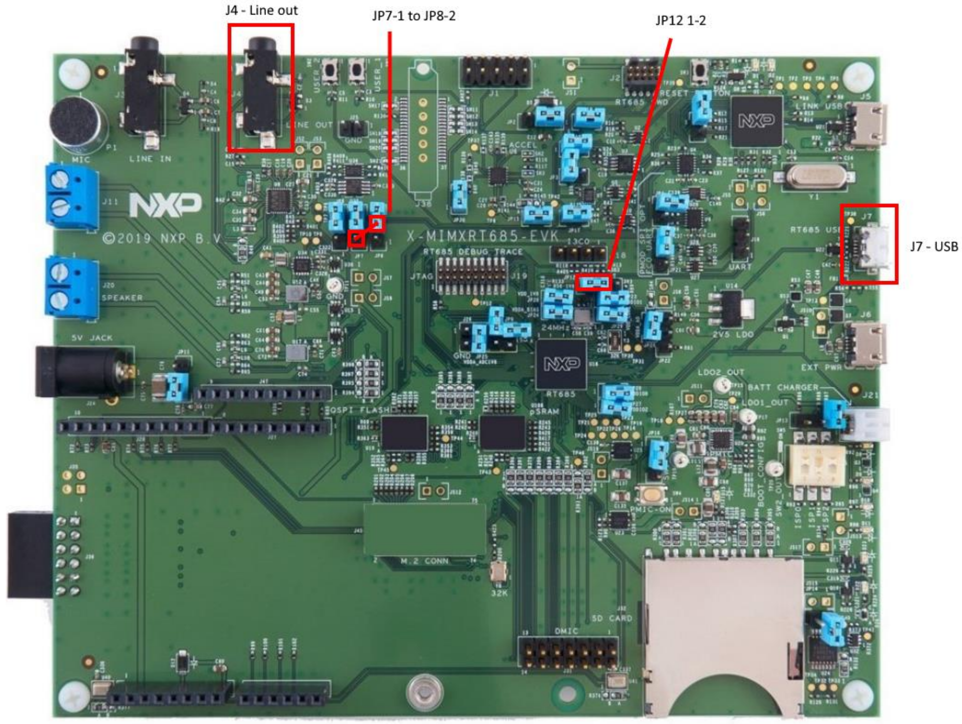
Audio Lab 3 – USB Audio + WM8904 Line-out

Lab description

Lab 3 is based on lab 1, the audio source is changed from the line-in of the codec to USB audio input from a USB Host with the USB Audio class device. The following figure shows a high-level diagram of the audio path.



MIMXRT685-EVK settings



- Connect PC to J7 with micro USB cable.
- Connect headphones with 3.5 mm audio jack to J4
- Ensure JP12 is connected at 1-2 (1.8V to VDDIO_1)
- Connect JP8-2 to JP7-1 with jumper wire (I2S TX)

HiFi4 project changes

In lab 1, the I²S from Flexcomm 1 was configured as RX, for lab 2 we will change this I²S interface to TX.

In "<hifi4 project>\codec\board_codec_i2s.c".

Uncomment line 20:

```
#define SWAP_I2S_TX_RX 1
```

This will have the following changes:

- Change the I2S_TX instance to I2S1.
- DMA request from Flexcomm interface 1.
- Attach audio PLL clock to Flexcomm 1.
- I²S instance as master.
- Adjust I²S clock divider for the desired sample rate. (see 25.7.3 Data rates in RT600 UM)

In **I2SRxCallback()** comment lines 89-94:

```
void I2SRxCallback(I2S_Type *base, i2s_dma_handle_t *handle,
          status_t completionStatus, void *userData) {
      AudioRxDMA_Queue_Element element;
      s_RxTransfer.dataSize = AUDIO_RX_TRANSFER_SIZE * sizeof(int16_t);
      s_RxTransfer.data = &audioRecDMABuff[AUDIO_RX_TRANSFER_SIZE
                  * audio_rx_buffer_index];

//    I2S_RxTransferReceiveDMA(I2S_RX, &s_RxHandle, s_RxTransfer);
//
//    element.pBufAddress = &audioRecDMABuff[AUDIO_RX_TRANSFER_SIZE
//                * audio_rx_buffer_index];
//    element.uBufSize = AUDIO_RX_TRANSFER_SIZE * sizeof(int16_t); /*
uBufSize in Bytes */
//    xos_msgq_put(p_audio_rx_dma_queue, (uint32_t *) &element);
      audio_rx_buffer_index ^= 1;
}
```

This part of the code from lab 1 was to update the audio buffer with the data from the line-in (I²S RX). When this code is commented, the audio buffer in *element.pBufAddress* will be updated with the USB buffer located in shared RAM that the CM33 is updating.

Reference code

The changes from the previous step can also be enabled by changing the build configuration in the Hifi4 project as follow:

- Right click “board_codec_i2s.c”, then choose “**Build->Exclude**”
- Right click “board_codec_dmic.c”, then choose “**Build->Exclude**”
- Right click “board_codec_arm.c”, then choose “**Build->Include**”

Build HiFi4 and ARM project

1. Rebuild the HiFi4 project. See lab 1 for more detailed instructions.
2. Copy the output binary files “dsp_text_release.bin” and “dsp_data_release.bin” from the HiFi4 project located at *“<workspace>\<project>\bin\rt600_sram2019_1\Release\”* and replace the files located in the ARM project at “<workspace>\evkmimxrt685_audio_bypass_cm33\”.
3. Clean the ARM project to ensure that the new HiFi4 output binary files are considered.
4. Rebuild ARM project. See lab 1 for more detailed instructions.

Run the ARM project

1. Debug and run the arm project.
2. Playback an audio file at PC side, Select USB audio speaker.
3. You should hear the music from the Headphones connected at J4.